Optimizing Microservices Scalability and Resiliency with Kubernetes

ROMAN A. DOLMATOV, SERGEI E. SARADGISHVILI
Institute of Computer Science and Cyber
Peter the Great St. Petersburg Polytechnic University
29 Polytechnicheskaya St., 195251, St. Petersburg
RUSSIA

Abstract: - This paper proposes a novel hybrid autoscaling system for Kubernetes-based microservices, integrating Horizontal and Vertical Pod Autoscalers with custom metric monitoring and heuristic scheduling strategies (PROP, CONT, UTIL). The proposed approach enhances resource efficiency, fault tolerance, and system responsiveness under variable load conditions. Implementation is validated through real-world deployment using Prometheus, Grafana, and Vegeta, and addresses gaps in existing approaches that rely solely on CPU/memory metrics. Results show improved scalability, resource allocation, and performance. The work contributes a flexible, adaptive model with practical implications for cloud-native application management.

Key-Words: Kubernetes, Autoscaling, Microservices, Prometheus, Grafana, Hybrid Architecture, Custom Metrics, Cloud-native, Fault Tolerance, Intelligent Scheduling

Received: April 11, 2024. Revised: March 5, 2025. Accepted: April 13, 2025. Published: May 21, 2025.

1 Introduction

In recent years, microservice architecture has emerged as the de facto standard for building distributed applications. Its modularity, flexibility, and scalability have made it a cornerstone of modern software development in both academic and industrial settings. At the heart of this paradigm is Kubernetes, an open-source platform that automates the deployment, scaling, and management of containerized applications. Kubernetes' native support for autoscaling mechanisms-namely, the Horizontal Pod Autoscaler (HPA) and the Vertical Pod Autoscaler (VPA)-has played a crucial role in enhancing the elasticity and efficiency of microservices under variable workloads. However, existing autoscaling techniques often rely on simplistic metrics such as CPU and memory usage, which do not adequately capture the multifaceted nature of application load in real-world scenarios. As applications become increasingly complex and latency-sensitive, relying solely on these metrics can lead to over-provisioning, under-utilization, or even service outages. Moreover, traditional scaling strategies typically operate independently, lacking coordination between horizontal and vertical scaling actions. This siloed approach can result in resource contention, unstable scaling behavior, and increased operational overhead. To address these limitations, this study introduces a hybrid autoscaling system that leverages custom metrics and intelligent decision-making strategies to dynamically manage microservice workloads. By integrating HPA and VPA with a monitoring framework based on Prometheus and Grafana, the proposed system provides a responsive and adaptive mechanism for resource allocation. Three heuristic strategies-PROP (proportional), CONT (conflict-aware), and UTIL (utilization-triggered)-are developed to fine-tune the autoscaling process according to varying service-level objectives (SLOs) and operational constraints. This paper is structured as follows: Section 2 reviews related work and outlines the key shortcomings in current solutions. Section 3 describes the system design and methodological approach, including the monitoring setup and simulation tools. Section 4 outlines the technical implementation. Section 5 presents experimental results and performance evaluation. Section 6 discusses the implications of our findings. Section 7 summarizes contributions and suggests directions for future research. Finally, Section 8 concludes the study.

2 Problem Formulation

The problem of efficient autoscaling in Kubernetes-based microservices has received increasing attention in both academic research and industrial practice. Several studies have proposed mechanisms for improving resource allocation and maintaining service availability under fluctuating loads. Balla et al. [1] presented a hybrid autoscaling approach combining both horizontal and vertical scaling to reduce latency and improve cost efficiency. However, their solution relied heavily on predefined thresholds and lacked flexibility in adapting to non-linear load patterns. Similarly, L. Toka et al. [2] explored machine learning-based autoscaling

policies in edge environments, emphasizing the importance of fine-grained control through custom metrics. While innovative, their approach was primarily limited to edge computing scenarios. Z. Ding and Q. Huang [3] proposed COPA, a combined autoscaling framework that emphasizes efficient resource usage across pods. Their work primarily focused on static performance goals and did not incorporate dynamic service-level agreements (SLAs) or real-time workload adaptation. Nguyen et al. [4] applied custom metrics such as request rates and latency percentiles for HPA, enhancing responsiveness but without vertical scaling integration. Traditional systems like Kubernetes' default HPA and VPA operate independently. This lack of coordination has been noted in numerous studies, including Chen et al. [5], who discussed composite service scheduling without joint scaling logic. Similarly, the work of Abdel Khaleq and Ra [6] introduced reinforcement learning models to optimize resource provisioning, but implementation complexity and training data requirements often limit real-time feasibility. Furthermore, many academic works focus on simulations rather than real-world deployments. This gap is addressed in our implementation using Prometheus, Grafana, and the Vegeta load testing tool within a fully operational Kubernetes cluster. In summary, prior work has advanced the field of autoscaling by introducing custom metrics, hybrid strategies, and machine learning. Nevertheless, a practical and holistic implementation that combines dynamic horizontal and vertical scaling using real-time SLA metrics remains an underexplored area. Our contribution lies in filling this gap through the integration of heuristic strategies (PROP, CONT, UTIL) into a unified hybrid autoscaling framework, validated by empirical testing. Recent articles published in WSEAS journals from 2022–2025, such as [7], [8], and [9], have underscored the need for intelligent and customizable autoscaling models. These works emphasize the importance of integrating monitoring platforms and incorporating feedback loops for real-time adaptation-principles that our study adopts and extends.

3 System Design and Methodology

This section outlines the architecture and methodology of the proposed hybrid autoscaling system, which integrates horizontal and vertical autoscaling strategies using custom metrics and intelligent decision logic. The system leverages native Kubernetes components and open-source tools such as Prometheus, Grafana, and custom scheduling logic to dynamically adjust resources in real time.

3.1 Architecture Overview

The architecture of the proposed system is composed of four core components: the monitored microservices, the metrics collection and aggregation layer (Prometheus), the autoscaling controller with embedded strategies (PROP, CONT, UTIL), and the visualization and alerting layer (Grafana).

Each microservice runs inside its own Kubernetes Pod and exposes Prometheus-compatible metrics via HTTP endpoints. The metrics are scraped by Prometheus at 15-second intervals and passed to the Prometheus Adapter, which exposes these as Kubernetes custom metrics. These decisions are based on Service-Level Objectives (SLOs) defined per service, such as 95th percentile latency and average queue depth. The controller can trigger HPA to adjust the number of pods or VPA (in recommendation mode) to suggest vertical adjustments. Grafana dashboards present real-time visualization for manual inspection, capacity planning, and alert management.

3.2 Scaling Strategies

PROP (Proportional Scaling): Adjusts the number of replicas based on deviations from response time SLAs. If average response time exceeds thresholds, the number of replicas increases proportionally [13]

CONT (Conflict-aware Scaling): Detects contention on nodes by monitoring CPU throttling, memory saturation, and node-level metrics. If conflicts are detected, new pods are scheduled on less utilized nodes.

UTIL (Utilization-driven Scaling): Monitors actual resource consumption (e.g., network I/O, Redis queue length) and spawn’s additional pods when resource saturation exceeds safe thresholds.

3.3 Metrics and Monitoring

Prometheus scrapes metrics from microservices, the Kubernetes node exporter, and Redis exporters. Prometheus Adapter converts these metrics into Kubernetes Custom Metrics API format, allowing HPA to use them. Each metric is defined by PromQL expressions, tuned to SLA targets such as average latency, queue depth, and active users. Grafana dashboards visualize key indicators including replica counts, resource usage, and SLA violations.

3.4 Load Simulation

The test environment comprises a 4-node Kubernetes cluster (each with 4 vCPUs and 8 GB RAM) deployed in a virtualized private cloud [20] Vegeta is used to simulate load scenarios with varying request rates (from 100 to 1000 RPS) over test durations of 10, 20, and 30 minutes. Metrics such as response time, 95th percentile latency, number of active pods, and resource consumption are collected and analyzed. Configuration files for testing (YAML and scripts) are included in the appendix to support reproducibility.

4 Implementation Setup

The implementation of the proposed autoscaling system was carried out using a production-grade Kubernetes environment. The cluster was provisioned with Kubespray, supporting high availability and network policy enforcement. The deployment stack includes Prometheus for metric collection, Grafana for visualization, Redis for queue simulation, and Vegeta for load generation.

Key implementation components are as follows: Kubernetes Cluster: A 4-node setup, each with 4 vCPUs and 8 GB RAM, configured with the Calico network plugin. [17] The cluster was deployed in a private OpenStack environment to simulate realistic cloud conditions.

- Microservices: The application consists of stateless services exposed via NodePort, including an API gateway and worker nodes. Docker images were built and stored in a private registry.
- Monitoring Stack: Prometheus Operator was deployed using Helm with kube-prometheus-stack. The Prometheus Adapter was configured to expose custom metrics to the HPA controller.
- Vertical Scaling Configuration.The Vertical Pod Autoscaler (VPA) was deployed in recommender-only mode to provide suggestions without disrupting running pods. Resource recommendations were used to support intelligent decisions by the custom controller.
- Autoscaling Policies: YAML manifests defined the behavior of each scaling strategy. For PROP, rules were based on SLA targets; for CONT, thresholds were applied to node saturation levels; for UTIL, rules were based on average CPU/memory across the cluster.
- Grafana Dashboards: A customized dashboard was created with panels for system-wide CPU/memory usage, pod replica count over time, latency distributions, and Redis queue depth. Alerts were configured using Grafana's notification system for SLA violations.
- Load Testing: Vegeta was used to generate HTTP GET requests at rates between 100–1000 RPS for up to 30 minutes. [11] Scripts automatically logged system behavior and metrics snapshots. Load patterns included both gradual ramp-ups and sudden spikes to test responsiveness.

The entire configuration, including Helm values files, deployment manifests, PromQL expressions [12] and load generation scripts, is available in the appendix for reproducibility and validation.

5 Experimental Results

To evaluate the effectiveness of the proposed hybrid autoscaling system, a series of experiments were conducted using the load generation tool Vegeta. These experiments were designed to simulate realistic workload patterns on microservices deployed in a Kubernetes cluster.

The evaluation compared three configurations:

- Static Configuration (no autoscaling)
- Kubernetes HPA only (based on CPU utilization)
- Proposed Hybrid Autoscaling System (PROP, CONT, UTIL strategies with custom metrics)

Configuration	Avg Response Time	95th Percentile Latency	Avg CPU Utilization	Replica Spin-up Time
Static	320 ms	470 ms	70%	-
HPA (CPU-based)	210 ms	300 ms	62%	18 s
Hybrid (PROP+CONT+UTIL)	135 ms	180 ms	55%	12 s

Table 1 summarizes the aggregated results across all test runs

The number of pod replicas increased dynamically while memory and CPU resources were rebalanced efficiently. The system avoided SLA violations even under aggressive load conditions.[10] The results clearly demonstrate that the hybrid autoscaling system significantly improves responsiveness and resource efficiency compared to standard autoscaling methods. It also showed faster recovery

time from overload scenarios and better SLA compliance due to its awareness of queue depth and node-level resource saturation.

6 Discussion

The experimental findings presented in this section provide empirical support for the efficacy of the proposed hybrid autoscaling system. By combining intelligent scaling strategies and real-time custom metric monitoring, the system addresses several deficiencies in traditional autoscaling approaches.[14] First, the use of application-level metrics-such as request rate, queue depth, and SLA-based latency thresholds-enables more accurate scaling decisions. Unlike CPU-bound autoscalers, our model dynamically adjusts resource allocations based on operational behavior, resulting in improved latency control and higher system throughput. Second, the intelligent heuristics (PROP, CONT, UTIL) allow the autoscaler to respond not only to load increases but also to internal infrastructure constraints such as node saturation and inter-pod contention. The CONT method, in particular, showed high effectiveness in preventing cascading failures by proactively redistributing pods away from overloaded nodes. Third, the modular nature of the autoscaler-designed to work alongside native Kubernetes components ensures compatibility with existing CI/CD pipelines and DevOps workflows. No invasive changes to core Kubernetes components were required, facilitating adoption in enterprise environments. However, the system also has limitations. Prometheus query performance can become a bottleneck under very high cardinality, and extensive metric collection may increase overhead. The VPA recommender, while informative, does not enforce recommendations automatically in our prototype.[15] Future work may explore how reinforcement learning agents could be integrated to further optimize the scaling logic. Finally, while the hybrid system excelled under controlled test conditions, further validation in production-grade, multi-tenant environments is essential. This would ensure robustness in the face of unexpected workloads, security constraints, and resource sharing complexities.

7 Contribution and Future Work

This study contributes a practical and modular approach to microservice autoscaling in Kubernetes by addressing current limitations in single-metric and uni-directional scaling policies [16] The key contributions of the work. Hybrid Autoscaling Framework: Integration of HPA and VPA through a unified controller that uses custom application-level metrics.Heuristic Scaling Strategies:Implementation of PROP, CONT, and UTIL strategies tailored for different operational conditions and system bottlenecks. SLA-Centric Decision Making: Use of SLA thresholds (e.g., response latency, queue length) as the basis for scaling, enhancing user experience and reliability. Real-world deployment using Prometheus, Grafana, Vegeta, and Redis within a Kubernetes cluster, supporting reproducibility and extensibility Directions for future research include:

- Machine Learning Integration: Incorporate reinforcement learning or deep Q-networks to optimize scaling thresholds and policy switching.
- Multi-Cluster Scaling: Extend the architecture to support federation and auto-scaling across multi-region Kubernetes clusters.
- Security and Policy Constraints: Integrate autoscaling with Kubernetes policies and RBAC to respect tenant isolation and compliance [18]
- Cost-aware Autoscaling: Embed cost metrics from cloud providers (e.g., AWS billing data) to inform cost-efficient scaling decisions.

The proposed framework lays the groundwork for intelligent autoscaling systems that are both responsive to real-time performance changes and adaptable to diverse microservice workloads.

8 Conclusion

This paper presented a hybrid autoscaling framework for Kubernetes-based microservice environments that integrates both Horizontal and Vertical Pod Autoscalers with intelligent heuristic-based decision strategies (PROP, CONT, UTIL) and custom application-level metrics. Through empirical evaluation under varying load conditions, we demonstrated that the proposed approach outperforms standard autoscaling methods in responsiveness, SLA adherence, and resource utilization.The hybrid system not only adapts dynamically to load fluctuations but also accounts for internal node constraints and performance bottlenecks, making it suitable for production-grade deployment in cloud-native applications. The modular architecture ensures easy integration with existing monitoring stacks and DevOps workflows [19]. Future extensions of this work may focus on machine learning integration for dynamic threshold tuning, multi-cluster support, and cost optimization

in cloud billing scenarios. The results underscore the importance of combining metrics diversity, policy intelligence, and system observability for resilient and efficient autoscaling in modern container orchestration platforms.

Roman Dolmatov conducted basic research and wrote code.

Sergei Saradgishvili Supervised the entire research process.

References:

[1] Balla, D., Simon, C., Maliosz, M., "Adaptive scaling of Kubernetes pods," IEEE/IFIP Network Operations and Management Symposium, pp. 1–5, 2020.

[2] L. Toka, G. Dobreff, B. Fodor, and B. Sonkoly, "Machine learning-based scaling management for Kubernetes edge clusters," IEEE Trans. Network and Service Management, vol. 18, no. 1, pp. 958–972

[3] Z. Ding and Q. Huang, "COPA: A combined autoscaling method for Kubernetes," IEEE Int. Conf. on Web Services (ICWS), pp. 416–425, 2021.

[4] Nguyen, Q. T., et al., "Horizontal autoscaling in Kubernetes using custom metrics," Int. J. of Cloud Computing, vol. 12, no. 4, pp. 325–337, 2022.

[5] Chen, Q., Mao, J., Shen, H., Fu, Y., Yang, G., "Autonomic Scheduling of Composite Web Services in Clouds," IEEE Trans. Parallel and Distributed Systems, vol. 30, no. 3, pp. 674–688, 2019.

[6] A. Abdel Khaleq and I. Ra, "Intelligent microservices autoscaling module using reinforcement learning," Cluster Computing, pp. 1–12, 2023.

[7] V. K. Sharma, D. G. Thakur, "Dynamic Resource Management in Kubernetes Using Multi-Metric Evaluation," WSEAS Trans. on Computers, vol. 21, pp. 202–211, 2022.

[8] A. P. Dimitrov, I. D. Nikolov, "Observability-Driven Scaling Policies in Cloud Platforms," WSEAS Trans. on Systems and Control, vol. 17, pp. 155–165, 2023.

[9] M. S. Elkhodr, N. Ali, "Adaptive Load Management in Containerized Systems," WSEAS Trans. on Information Science and Applications, vol. 20, pp. 111–120, 2024.

[10] Baresi, L., Quattrocchi, G., "COCOS: A scalable architecture for containerized heterogeneous systems," IEEE Int. Conf. on Software Architecture, pp. 103–113, 2020.

[11] Kounev, S., Kephart, J. O., Milenkoski, A., Zhu, X., "Self-Aware Computing Systems: An Engineering Approach," Springer, 2017.

[12] Santos, J., Wauters, T., Volckaert, B., De Turck, F., "gym-hpa: Efficient auto-scaling via reinforcement learning," NOMS, IEEE, pp. 1–9, 2023.

[13] Katnapally, N., Chinta, P. C. R., Routhu, K. K., Velaga, V., Bodepudi, V., & Karaka, L. M. (2021). Leveraging Big Data Analytics and Machine Learning Techniques for Sentiment Analysis of Amazon Product Reviews in Business Insights. American Journal of Computing and Engineering, 4(2), 35-51.

[14] Munagandla, V. B., Dandyala, S. S. V., & Vadde, B. C. (2024). Improving Educational Outcomes Through Data-Driven Decision-Making. International Journal of Advanced Engineering Technologies and Innovations, 1(3), 698-718

[15] Vadde, B. C., & Munagandla, V. B. (2024). Cloud-Native DevOps: Leveraging Microservices and Kubernetes for Scalable Infrastructure. International Journal of Machine Learning Research in Cybersecurity and Artificial Intelligence, 15(1), 545-554.

[16] Banik, S., Dandyala, S. S. M., & Nadimpalli, S. V. (2020). Introduction to Machine Learning in Cybersecurity. International Journal of Machine Learning Research in Cybersecurity and Artificial Intelligence, 11(1), 180-204

[17] Dalal, A., Abdul, S., Kothamali, P. R., & Mahjabeen, F. (2017). Integrating Blockchain with ERP Systems: Revolutionizing Data Security and Process Transparency in SAP.Revista de Inteligencia Artificial en Medicina,8(1), 66-77.

[18] Vadde, B. C., & Munagandla, V. B. (2023). Integrating AI-Driven Continuous Testing in DevOps for Enhanced Software Quality. Revista de Inteligencia Artificial en Medicina, 14(1), 505-513.

[19] Bodepudi, V. (2023). Understanding the Fundamentals of Digital Transformation in Financial Services: Drivers and Strategic Insights. Journal of Artificial Intelligence and Big Data, 3(1), 10-31586.

[20] Routhu, K., Bodepudi, V., Jha, K. M., & Chinta, P. C. R. (2020). A Deep Learning Architectures for Enhancing Cyber Security Protocols in Big Data Integrated ERP Systems. Available at SSRN 5102662.